Epidemiology Group

Software Testing

We had a heavy focus on component testing and interface testing for our project such as, pathfinding, infection rate, collision, player movement, and how these things interacted with each other. We tested the pathfinding system by creating a large list of locations that the AI could move between, and then verifying that the AI looped from one location to the other. Later this was expanded upon with a fully fledged destination system algorithm. We intentionally set-up many defect testing scenarios in which we would run the player into walls in which colliders were set up, forcibly spawn AI in undesirable but plausible locations, and have the player set-up oddly placed infection objects. This at first yielded errors as some AI would not continue their pathfinding loop, however we were able to correct these use a version of incremental development on that component, and then verify that the system worked properly. We tested the infection rate of the AI in a similar manner; at first we started with a small amount of AI, and ensured that they could pass their infected state to each other, and then we verified that this still worked for large amounts of AI. As a natural flow from increasing data by means of testing with many AI, this ended up being a form of stress testing by testing the performance of the system. One result of this stress testing had us cohesively come to the conclusion that our system performed very well under failure conditions; we expected the system to create frame rate issues and possibly crash when too many computations were performed every frame. However, our stress tests revealed a heartening amount of resilience.

We used an automated testing system in the form of console debugging built into the Unity Editor. And even though our simulation is mostly automated there are player actions that are performed and these actions could have been written as a test program. This would have been a lot of work at first, but then it could have sped up by testing user functions. Although

extensive user testing was not completed due to time constraints and availability of software distribution builds, we did do some user testing in the form of opinionated testing under acquaintances and classmates. By releasing it to others outside of these bounds we could have seen what others would try to do inside of the game such as trying to cheat or finding bugs that we could have addressed. The bugs could have been addressed along with changing gameplay to make it more challenging to cheat. This is similar to when publishers on Steam and other game distribution software allows early access to their games so people can see what the game is like and allows the developers to see what they need to fix. In accordance, one of the obvious ups to open software distribution and marketing is the ability to produce Alpha and Beta tests. This can be seen as a good or bad thing since releasing a game early can color an impression of the game and change how people will view it. If we had more time to flesh this out, releasing this to the public might have been an option that we could consider; however, at this point we do not think what we have is ready to be released.

In conclusion, there are many components of testing that were utilized in our project over the past several months. We used variations of every form of testing; unit testing, component testing, and system testing. In particular, because we used a game engine in which the software is built on top of mechanic logic, we were able to do extensive system testing interlocked with much of our unit and component testing. This allowed us to validate and verify that the software performed to the specifications we set forth very early on, while at the same time setting goals to improve in the long run. On top of this, as we were able to do so much extensive system testing, it made a test driven approach very feasible. We could test the whole system, write functionality, and then test the whole system on the fly. We wrote tests to test functionality along the way, and this made it extremely easy to find bugs involving things such as when AI would sometimes deallocate destinations due to a logical programming error. From there, we were

able to dive right into a state of component testing, then switch immediately back to system testing to make sure every component interacted as expected after this error was tested for existence extensively. All-in-all, game development is a variable, challenging, and rewarding environment for learning the ins and outs of testing code bases.